Cellular Network Infrastructure - Feature #4073

try to get counter and vty reference information at compile time

06/21/2019 02:55 PM - laforge

Status:	Stalled	**Start date:**	06/21/2019
Priority:	Low	**Due date:**	
Assignee:	Hoernchen	**% Done:**	70%
Category:			
Target version:			
Spec Reference:			

Description

The existing approach of extracting counter and vty information at runtime has various disadvantages. We'd like to explore options to generate this information from the source code. Whether that's using static code analysis, an existing C parser of some form, clang, ... doesn't really matter.

The goal is to generate the same textual (asciidoc for counters, xml for VTY) format as the existing 'runtime' approach.

Related issues:

Related to Cellular Network Infrastructure - Feature #4044: regenerate vty re...	**Stalled**	**06/04/2019**

History

#1 - 06/28/2019 07:39 AM - osmith

- Related to Feature #4044: regenerate vty reference during release process added

#2 - 06/28/2019 07:50 AM - osmith

I prefer this approach to the solution we arrived in #4044 (enforce generating the documentation with each commit in jenkins' gerrit-verfication job), if it is feasible.

@Hoernchen, you have explained in detail how the issue is going yesterday on the phone. Can you post a quick summary here too? Thanks!

#3 - 07/02/2019 09:21 AM - Hoernchen

We want to extract this from the code, so the only reasonable choice is Clang/LLVM, more specifically LibTooling, because LibClang offers a stable API, but accessing the AST through a C interface is cumbersome. Looking for the known struct names should suffice, there is also the possibility of attaching annotations to functions and variables to augment this approach.
It's also possible to modify all build scripts to support compiling and linking everything into one large bitcode blob, and the optimizing away all unused code in order to produce one large bitcode blob that contains all vty commands (including those provided by the used libs) that can be used, but I don't think that is necessary at this point.

#4 - 07/03/2019 12:20 AM - laforge

On Tue, Jul 02, 2019 at 09:21:21AM +0000, Hoernchen [REDMINE] wrote:

> It's also possible to modify all build scripts to support compiling and linking everything into one large bitcode blob, and the optimizing away all unused code in order to produce one large bitcode blob that contains all vty commands (including those provided by the used libs) that can be used, but I don't think that is necessary at this point.

the Osmocom user manuals include the VTY reference of the libraries used, so if I understand the above correctly, it sounds like we need to do this 'linking everything into one bitcode blob' approach?

Alternatively, one could of course generate the reference of each library as separate xml/asciidoc and then merge it at that point. The problem with any of these approaches is that there are tons of things in libraries which are not actually used.

So to get the "proper" output, we would have to look at only those VTY commands and counters which are actually registered (used) by the given program.

#5 - 12/01/2019 09:35 AM - laforge

Please update the ticket with a description of the latest status, and set the perceived percentage (and probably "stalled")

#6 - 12/02/2019 03:05 PM - Hoernchen

- Status changed from New to Stalled

- % Done changed from 0 to 70

The current solution consists of two parts, step one is linking everything into one big bitcode file, step two is optimizing this to get rid of unused code, and then getting the debug information for occurences of vty command registration calls and so on, which are then used to look up and parse the corresponding code using a compilaton database file.

The issue with this approach is that most projects do not feature nice single make targets for the various parts like libs/tests/whatever due to the atrocious libtool and just adding a nice makefile rule that allows outputting a nice compilation database (per file) is not possible either, so one way to circumvent the latter issue is by using a unix domain socket as "filename", since writes up to one page size are atomic this should work with concurrent compiler (clang) invocations, the other is to embed the bitcode into all object files and extract them afterwards, and then hand-craft a bit of find magic to get only the files that we need without tests or multiple main functions and stuff like that to make the linking work.

It looks like this:

```
CC=clang-8 CFLAGS="-O0 -g -fembed-bitcode" ./configure
#liboso-netif
make clean
bear make -j

find . -iname "*.o" | xargs -I{} objcopy -O binary --only-section=.llvmbc {} {}.bc

#make clean # remove all *.o
cat compile_commands.json | jq '.[] | select( .directory | contains("/src") )' | jq ' .arguments[0] |= "clang-8 -emit-llvm"' | jq -r '.arguments |= join(" ") |  "cd \(.directory) && \(.arguments)"' | xargs -d '\n' -n1 -P 4 sh -c
find . -iname "*.o" -print0 |xargs -0 -n1 file | grep LLVM |cut -d: -f1 | grep \.libs | xargs llvm-link-8 -o all.bc
```

So the actual issue is not so much extracting the information, but getting a nice compilation database per project/a sane subset of bitcode files without having to resort to brittle hand-crafted lists of grep and find magic for all projects that need to be updated all the time, due to the way the arcane build system works.

#7 - 01/08/2020 10:23 PM - laforge

- Priority changed from Normal to Low